Jack Girard
ECE 637
Lab 3: Neighborhoods and Connected Components

1. Area Fill

1. Print out of the image *img22gd2.tif*

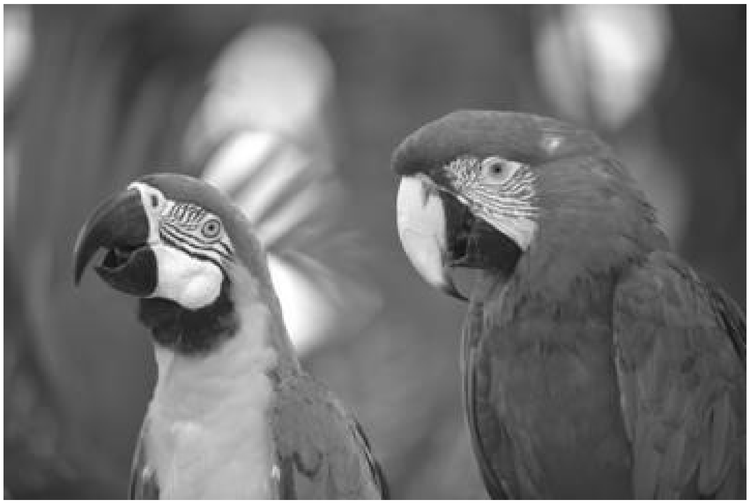

Figure 1: img22gd2.tif

2. Print out of the image showing the connected set for s = (67, 45), and T = 2

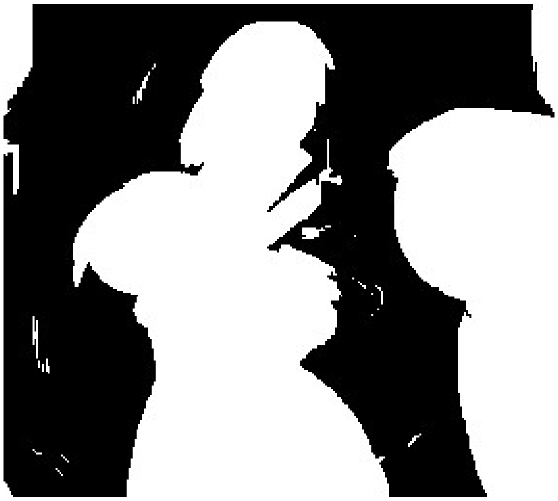

Figure 2: Connected Set for T = 2

3. Print out of the image showing the connected set for s = (67, 45), and T = 1

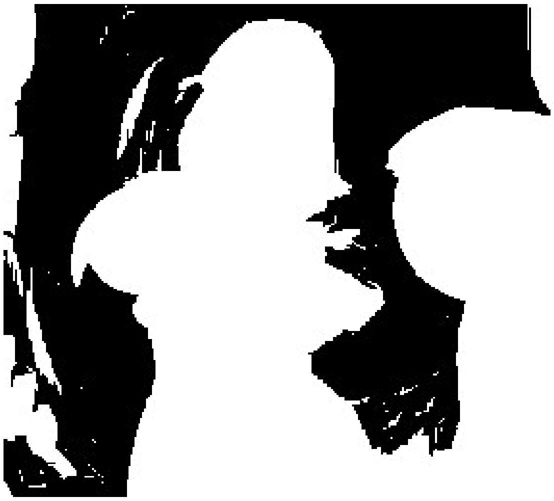

Figure 3: Connected Set for T = 1

4. Print out of the image showing the connected set for s = (67, 45), and T = 3

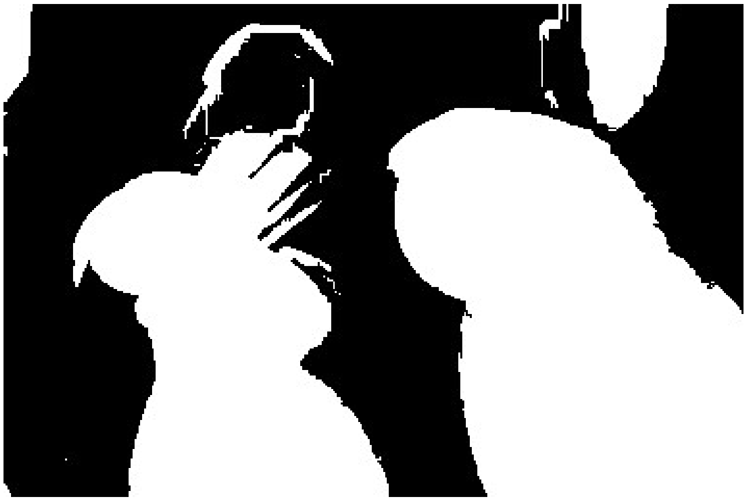

Figure 4: Connected Set for T = 3

5. Listing of C Code: **See Next Page**

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "defs.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  // define pixel of interest/seed pixel
  struct pixel s;
  s.m = 67;
  s.n = 45;
  // declare integers i and j for looping later on
  int i, j;
  // declare and/or initialize further variables needed
  int ClassLabel = 1;
  int NumConnectedPixels;
  double T = 3;

  if ( argc != 2 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be grayscale\n" );
    exit ( 1 );
  }

  // create seg array, initialize all values to zero to start
```

```c
unsigned int** seg = (unsigned int**)get_img(input_img.width, input_img.height, sizeof(unsigned int));
for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        seg[i][j] = 0;
    }
}

// call ConnectedSet() function to determine the connected neighbors to the seed pixel s
ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height, ClassLabel, seg, &NumConnectedPixels);

// display number of connected pixels
printf("%d", NumConnectedPixels);

// for each pixel in input_img.mono, assign either 0 or 255 depending on neighbor label within seg
for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        if (seg[i][j] == ClassLabel) {
            input_img.mono[i][j] = 0;
        }
        else {
            input_img.mono[i][j] = 255;
        }
    }
}

// generate new tiff of proper size, assign to input_img for writing out later
get_TIFF(&output_img, input_img.height, input_img.width, 'g');
output_img = input_img;

/* open color image file */
if ((fp = fopen("output.tif", "wb")) == NULL) {
    fprintf(stderr, "cannot open file out.tif\n");
    exit(1);
}

/* write color image */
if (write_TIFF(fp, &output_img)) {
    fprintf(stderr, "error writing TIFF file %s\n", argv[2]);
    exit(1);
}

/* close color image file */
fclose(fp);

```

```c
  /* de-allocate space which was used for the images */
  free_TIFF(&(input_img));
  free_img((void*)seg);

  return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise, \n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```

```c
#pragma once
#include <stdlib.h>

struct pixel {
    int m, n;       // m = row, n = col
};

void ConnectedNeighbors(struct pixel s, double T, unsigned char** img, int width, int height, int* M, struct pixel c[4]);

void ConnectedSet(struct pixel s, double T, unsigned char** img, int width, int height, int ClassLabel, unsigned int** seg, int* NumConPixels);
```

```c
#include "defs.h"

void ConnectedNeighbors(struct pixel s, double T, unsigned char** img, int width, int height, int* M, struct pixel c[4]) {
    *M = 0;
    if (s.m - 1 >= 0 && abs(img[s.m][s.n] - img[s.m - 1][s.n]) <= T) {
        c[*M].m = s.m - 1;
        c[*M].n = s.n;
        (*M)++;
    }
    if (s.m + 1 < height && abs(img[s.m][s.n] - img[s.m + 1][s.n]) <= T) {
        c[*M].m = s.m + 1;
        c[*M].n = s.n;
        (*M)++;
    }
    if (s.n - 1 >= 0 && abs(img[s.m][s.n] - img[s.m][s.n - 1]) <= T) {
        c[*M].m = s.m;
        c[*M].n = s.n - 1;
        (*M)++;
    } if (s.n + 1 < width && abs(img[s.m][s.n] - img[s.m][s.n + 1]) <= T) {
        c[*M].m = s.m;
        c[*M].n = s.n + 1;
        (*M)++;
    }
}
```

```c
#include "defs.h"
#ifdef _MSC_VER
#include <malloc.h>
#endif
#include <assert.h>

void ConnectedSet(struct pixel s, double T, unsigned char** img, int width, int height, int ClassLabel, unsigned int** seg, int* NumConPixels) {
    *NumConPixels = 0;
    struct pixel* B = alloca(sizeof(struct pixel) * width * height);
    unsigned int lenB = 0;
    unsigned int M = 0;
    struct pixel c[4];

    if (B == NULL)
        return;

    B[lenB++] = s;

    while (lenB > 0) {

        assert(lenB < width* height);
        s = B[lenB - 1];
        --lenB;

        ConnectedNeighbors(s, T, img, width, height, &M, c);

        if (seg[s.m][s.n] == 0) {
            (*NumConPixels)++;
        }

        seg[s.m][s.n] = ClassLabel;

        for (int i = 0; i < M; i++) {
            if (seg[c[i].m][c[i].n] == 0) {
                B[lenB++] = c[i];
            }
        }
    }
}
```

2. Image Segmentation

1. Print outs of the randomly colored segmentation for T = 1, T = 2, and T = 3

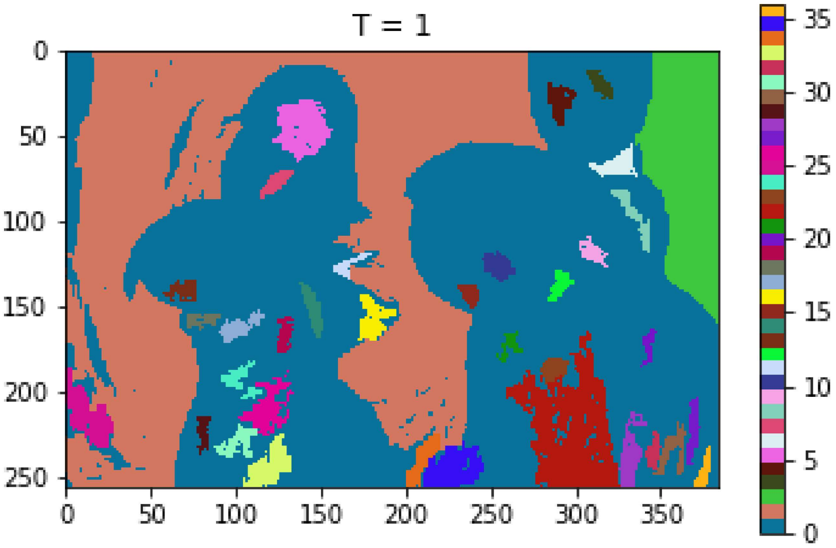

Figure 5: Colored Segmentation for T = 1

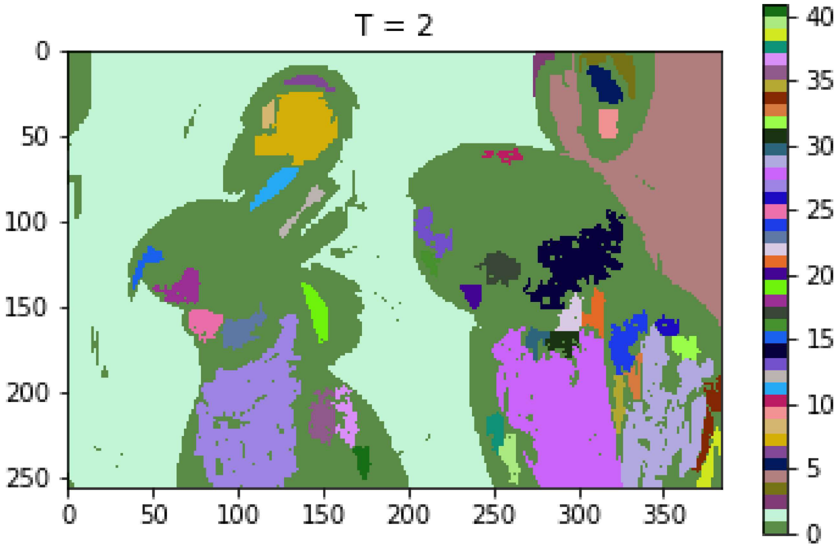

Figure 6: Colored Segmentation for T = 2

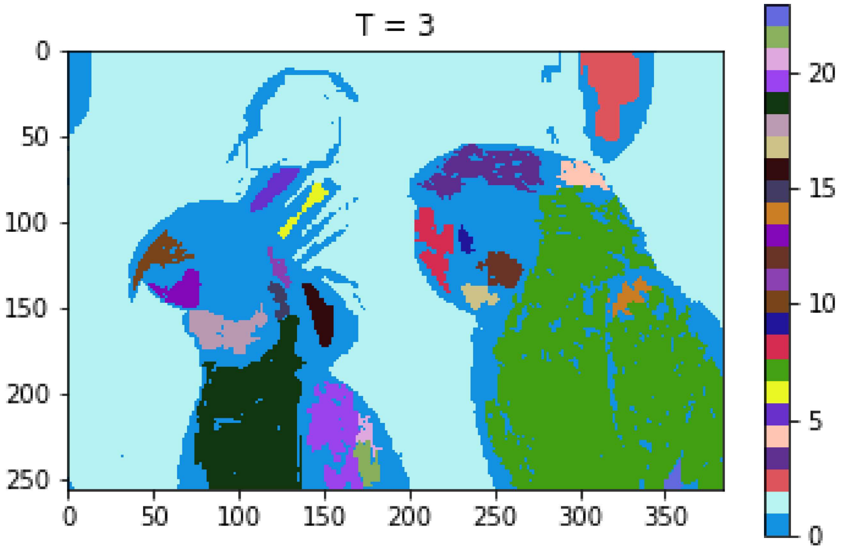

Figure 7: Colored Segmentation for T = 3

2. Listing of the number of regions generated for each of the values of T

Table 1: Pixel Counts

T Value	Region Count
1	36
2	41
3	23

3. Listing of C Code: **See Next Page**

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "defs.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, output_img;
  // define pixel of interest/seed pixel
  struct pixel s;
  s.m = 67;
  s.n = 45;
  // declare integers i and j for looping later on
  int i, j;
  // declare and/or initialize further variables needed
  int ClassLabel = 1;
  int NumConnectedPixels;
  double T = 2;

  if ( argc != 2 ) error( argv[0] );

  /* open image file */
  if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
  }

  /* read image */
  if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
  }

  /* close image file */
  fclose ( fp );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be grayscale\n" );
    exit ( 1 );
  }

  // create seg array, initialize all values to zero to start
```

```c
    unsigned int** seg = (unsigned int**)get_img(input_img.width, input_img.height, sizeof(unsigned int));
    for (i = 0; i < input_img.height; i++) {
        for (j = 0; j < input_img.width; j++) {
            seg[i][j] = 0;
        }
    }

    // declare variable to increment and count large connected sets with, initialize to 2
    int labelCount = 2;
    // declare integer to store the number of connected pixels for each ConnectedSets() execution
    int numConnections;

    // for eac pixel in image
    for (i = 0; i < input_img.height; i++) {
        for (j = 0; j < input_img.width; j++) {
            // if pixel has not been checked
            if (seg[i][j] == 0) {
                s.m = i;
                s.n = j;
                ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height, labelCount, seg, &numConnections);
                // if connected set qualifies for a large connected set
                if (numConnections > 100) {
                    labelCount++;
                }
                // otherwise, run ConnectedSet() with ClassLabel = 1 to keep track of small connected sets
                // (NOTE: this will be different from the label count, which was initialized to 2 and incremented thereafter
                else {
                    ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height, ClassLabel, seg, &numConnections);
                }
            }
        }
    }

    // display number of large connected set regions
    printf("Number of regions generated for %1f is %d \n", T, labelCount - 2);

    // for each pixel
    for (i = 0; i < input_img.height; i++) {
        for (j = 0; j < input_img.width; j++) {
            // decrement seg so all small connected sets are set to zero
            seg[i][j] = seg[i][j] - 1;
```

```c
          // assign input_img.mono to seg
          input_img.mono[i][j] = seg[i][j];
      }
  }

  // Get new tiff object of correct size, assign it to input_img
  get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );
  output_img = input_img;

  /* open color image file */
  if ( ( fp = fopen ( "segmentation.tif", "wb" ) ) == NULL ) {
      fprintf ( stderr, "cannot open file out.tif\n");
      exit ( 1 );
  }

  /* write color image */
  if ( write_TIFF ( fp, &output_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
      exit ( 1 );
  }

  /* close color image file */
  fclose(fp);

  /* de-allocate space which was used for the images */
  free_TIFF(&(input_img));
  free_img((void*)seg);

  return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds noise, \n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```